Lab 2:

Structured Program Development in C

(Part A: Your first C programs - integers, arithmetic, decision making, Part B: basic problem-solving techniques, formulating algorithms)

Learning Objectives

1. To be able to use arithmetic operators.
2. To understand the precedence of arithmetic operators.
3. To be able to write simple decision-making statements.
4. To understand basic problem-solving techniques.
5. To be able to develop algorithms through the process of top-down, stepwise refinement.
6. To be able to use the `if` selection statement and `if…else` selection statement to select actions.
7. To be able to use the while repetition statement to execute statements in a program repeatedly.
8. To understand structured programming.
9. To be able to use the increment, decrement and assignment operators.

Lab Work

General Note: Study every exercise and for each of them draw a structure diagram. Type in all these programs and run them inside your C development environment.

In your log-book take note of all the results from the execution of these programs and write down any problems that you had to solve.

For each program of the exercises, explain clearly what each section of the program is doing. You may need to perform a walk-through, line by line, where this is appropriate. Record all your explanations in the log-book.

Remember to put a date in your log-book and also to offer some discussions and a brief conclusion after every week's work.

Finally, complete the programming assignments at the end of the exercises, then enter comments in your logbook as you are doing this and save to your floppy disk for submission with your logbook.

Part A: Your first C programs - integers, arithmetic, decision making

The next C program uses the standard library function `scanf` to obtain two integers typed by a user at the keyboard, computes the sum of these values and prints the result using `printf`.

Exercise 1

1. Analyze the structure of the following C program.
2. Understand the program and predict the outcome in your logbook.

```
/* Fig. 2.5: fig02_05.c
Addition program */

#include <stdio.h>

/* function main begins program execution */
```

```
int main()
{
   int integer1; /* first number to be input by user  */
   int integer2; /* second number to be input by user */
   int sum;      /* variable in which sum will be stored */

   printf( "Enter first integer\n" ); /* prompt */
   scanf( "%d", &integer1 );          /* read an integer */

   printf( "Enter second integer\n" ); /* prompt */
   scanf( "%d", &integer2 );           /* read an integer */

   sum = integer1 + integer2; /* assign total to sum */

   printf( "Sum is %d\n", sum ); /* print sum */

   return 0; /* indicate that program ended successfully */

} /* end function main */
```

---

3. Create, compile and Run the program. Explain and analyze it in your logbook.

Exercise 2

1. Analyze the structure of the following four C programs.
2. Understand each program and predict the outcome in your logbook.

(i) The WHILE Loop

example of a "`while`" loop is given below.

```
main()
{
int count;
   count = 0;
   while (count < 6) {
      printf("The value of count is %d\n",count);
      count = count + 1;
   }
}
```

(ii) THE DO-WHILE LOOP

The following program shows an example of a `do-while` loop.

```
main()
{
int i;
   i = 0;
   do {
      printf("The value of i is now %d\n",i);
      i = i + 1;
   } while (i < 5);
}
```

(iii) THE FOR LOOP

The `for` loop is an alternative way to describe the `while` loop. The format is slightly different in that the condition that determines whether the loop is performed is more complicated. Here is an example.

```
main()
int index;
   for(index = 0;index < 6;index = index + 1)
     printf("The value of the index is %d\n",index);
}
```

(iv) THE IF STATEMENT

`If` statements are used to determine test conditions, in a manner quite similar to that discussed earlier for loops. An example of a program using the conditional `If` statement is shown below.

```
main()
{
int data;
   for(data = 0;data < 10;data = data + 1) {
      if (data == 2) // if statement 1
         printf("Data is now equal to %d\n",data);
      if (data < 5) // if statement 2
         printf("Data is now %d, which is less than 5\n",data);
      else // else if statement2 in not is satisfied
         printf("Data is now %d, which is greater than 4\n",data);
   }  /* end of for loop */
}
         printf("Data is now equal to %d\n",data);
```

3. Create, compile and Run the program. Explain and analyze it in your logbook.

The next program uses six `if` statements to compare two numbers input by the user. If the condition in any of these if statements is satisfied, the `printf` statement associated with that `if` is executed.

Exercise 3

1. Analyze the structure of the following C program.
2. Understand the program and predict the outcome in your logbook.

```
/* Fig. 2.13: fig02_13.c
   Using if statements, relational
   operators, and equality operators */

#include <stdio.h>

/* function main begins program execution */
main()
{
   int num1; /* first number to be read from user  */
   int num2; /* second number to be read from user */

   printf( "Enter two integers, and I will tell you\n" );
   printf( "the relationships they satisfy: " );

   scanf( "%d%d", &num1, &num2 ); /* read two integers */

   if ( num1 == num2 ) {
      printf( "%d is equal to %d\n", num1, num2 );
   } /* end if */

   if ( num1 != num2 ) {
      printf( "%d is not equal to %d\n", num1, num2 );
   } /* end if */
```

```
   if ( num1 < num2 ) {
      printf( "%d is less than %d\n", num1, num2 );
   } /* end if */

   if ( num1 > num2 ) {
      printf( "%d is greater than %d\n", num1, num2 );
   } /* end if */

   if ( num1 <= num2 ) {
      printf( "%d is less than or equal to %d\n", num1, num2 );
   } /* end if */

   if ( num1 >= num2 ) {
      printf( "%d is greater than or equal to %d\n", num1, num2 );
   } /* end if */

   return 0; /* indicate that program ended successfully */

} /* end function main */
```

3. Create, compile and Run the program. Explain and analyse it in your logbook.

The next program uses six `if` statements to compare two numbers input by the user. If the condition in any of these if statements is satisfied, the `printf` statement associated with that `if` is executed.

Part B: basic problem-solving techniques, formulating algorithms

Problem: Determine the class average of a class of ten students with grade integers in the rage of 0 to 100. Your algorithm makes use of counter-controlled repetition.

Exercise 1

1. Analyze the structure of the following C program.
2. Understand the program and predict the outcome in your logbook.
3. Provide a pseudocode algorithm to solve the class average problem.

```
/* Fig. 3.6: fig03_06.c
   Class average program with counter-controlled repetition */

#include <stdio.h>

/* function main begins program execution */
int main()
{
   int counter; /* number of grade to be entered next */
   int grade;   /* grade value */
   int total;   /* sum of grades input by user */
   int average; /* average of grades */

   /* initialization phase */
   total = 0;   /* initialize total */
   counter = 1; /* initialize loop counter */

   /* processing phase */
   while ( counter <= 10 ) {     /* loop 10 times */
      printf( "Enter grade: " ); /* prompt for input */
      scanf( "%d", &grade );     /* read grade from user */
      total = total + grade;     /* add grade to total */
      counter = counter + 1;     /* increment counter */
```

```
   } /* end while */

   /* termination phase */
   average = total / 10; /* integer division */

   printf( "Class average is %d\n", average ); /* display result */

   return 0; /* indicate program ended successfully */

} /* end function main */
```

4. Create, compile and Run the program. Explain and analyze it in your logbook.

Programming assignments

1. Write a program that writes your name on the monitor ten times. Write this program three times, once with each looping method as you learned in Exercise 2.

2. Write a program that reads in five integers and then determines and prints the largest and the smallest integers in the group.

3. Write a program that reads an integer and determines and prints whether it is odd or even. [Hint: Use the remainder operator. An even number is a multiple of two. Any multiple of two leaves a remainder of zero when divided by 2].